NeuroMem SDK
User's Manual

API INTERFACING TO A NEUROMEM SILICON NETWORK

Version 2.4
Revised 05/19/2015

NeuroMem SDK is a product of General Vision, Inc. (GV)

This manual is copyrighted and published by GV. All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of GV.

For information about ownership, copyrights, warranties and liabilities, refer to the document Standard Terms And Conditions Of Sale or contact us at www.general-vision.com.

TABLE OF CONTENTS

Table of Contents 3
Introduction 5
Bin Folder 5
Example folders 5
C/C++ 5
C# 5
Python 5
MatLab 5
Hardware Interface 6
int Version(); 6
Void SetPlatform(int platform); 6
int Connect(int DeviceID); 6
int Disconnect(); 6
int CountNeuronsandReset(); 6
Learning and Recognition functions 6
void Broadcast(byte vector[], int length) 6
void BroadcastSR(byte vector[], int length) 7
int Learn(unsigned char vector[], int length, int category); 7
int Recognize(unsigned char vector[], int length, int K, int distance[], int category[], int Nid[]); 7
void ReadLastVector(unsigned char vector[]); 7
Loading and Retrieval of neurons' content 8
int CommittedNeurons(); 8
int ClearNeurons(); 8
void ReadNeuron(int neuronID, unsigned char model[], int context, int aif, int minif, int category); 8
int ReadNeurons(unsigned char *neurons, int ncount); 8
int WriteNeurons(unsigned char *neurons, int ncount); 9
int SaveNeurons(char *filename); 9
int LoadNeurons(char *filename); 9
Vector Batch Processing 9
int LearnVectors(unsigned char *vectors, int vectNbr, int vectLen, int *categories, bool ResetBeforeLearning, bool iterative); 9
int BuildCodebook(unsigned char *vectors, int vectNbr, int vectLen, int CatAllocMode, int InitialCategory, bool WithoutUncertainty); 10
int Clusterize(unsigned char *vectors, int vectNbr, int vectLen, int CatAllocMode); 11
void RecognizeVectors(unsigned char *vectors, int vectNbr, int vectLen, int K, int *distances, int *categories, int *nids); 11
int MatchVectors(unsigned char *vectorsRef, int vectRefNbr, unsigned char *vectors, int vectNbr, int vectLen, int maxDistance, int *indexPairs, int *matchDistances); 12
GV communication protocol 12
int Write(unsigned char module, unsigned char reg, int data); 13
int Read(unsigned char module, unsigned char reg); 13
int Write_Addr(int addr, int length_inByte, unsigned char data[]); 13
int Read_Addr(int addr, int length_inByte, unsigned char data[]); 13
From Address Map to (Module,Register) 14
Cycle Accurate timings 14
void EnableClockCounter() 14
void DisableClockCounter() 14
int ReadClockCounter() 15
void ClearClockCounter() 15
Example using clock counter 15

Example script 15
Step 1: Learn 15
Step2: Recognition 16
Case of uncertainty, closer to Neuron#1 16
Case of uncertainty, equi-distant to Neuron#1 and Neuron#2 16
Case of uncertainty, closer to Neuron#2 16
Case of unknown 16
Step3: Adaptive learning 17
Supplement: Text Analytics functions 18
Remarks about patterns composed of ASCII characters 18
int BuildDictionnary(char *Filename, int NbrOfFrames, char *delimiter) 18
void SaveDictionnary(char *Filename) 19
int LoadDictionnary(char *Filename) 19
void ProcessPost(char* post, int* woCount) 19
void BuildScores(float* scoreTable) 19
void DisplayDictionnary() 19

INTRODUCTION

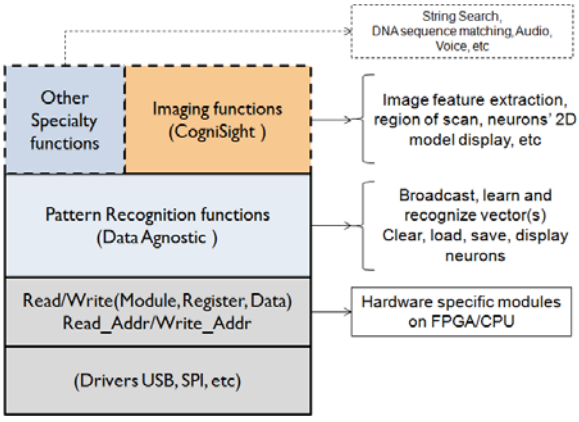
The NeuroMem SDK let's you take advantage of a NeuroMem hardware neural network to match and classify patterns at high speed with deterministic latency and using low-power appliances.

The NeuroMem SDK interfaces with the following hardware from General Vision

- Single NeuroStack board featuring a chain of 4096 neurons (default configuration)
- Stack of N NeuroStack boards featuring a chain of up to N*4096 neurons

The selection of the hardware is made at the instantiation of the class through the "platform" input parameter.

The functions supplied in the NeuroMem SDK are all based on Register Transfer Level (RTL) protocol accessing the NeuroMem CM1K chip. Higher level functions manipulating agnostic or specific data types are built on this protocol as shown in the following diagram. Depending on your hardware platform, this RTL protocol can also be used to address other components such as memory, on-board sensors, and more.


String Search, DNA sequence matching, Audio, Voice, etc
Other Specialty functions
Imaging functions (CogniSight)
Image feature extraction, region of scan, neurons' 2D model display, etc
Pattern Recognition functions (Data Agnostic)
Broadcast, learn and recognize vector(s) Clear, load, save, display neurons
Read/Write(Module, Register, Data) Read_Addr/Write_Addr
Hardware specific modules on FPGA/CPU
(Drivers USB, SPI, etc)

Bin Folder

- GVEngine.dll compiled for 32-bit and 64-bit operating systems
- GVAPI.h, header file with the declaration of the functions
- GVREG.h, file with the declaration of the CM1K registers

Example folders

C/C++

The Test_NM_CPlusPlus program calls the GVEngine.lib. The declaration of typedef and function calls at located in the GVAPI.h. The Test Neuron script is described in Appendix of this manual.

C#

The Test_NM_CSharp program calls the GVEngine.dll. The declaration of the function calls at located in the file GVAPI.cs. The Test Neuron script is described in Appendix of this manual.

PYTHON

The Test_NM_Python27 programs call the GVEngine.dll through a "ctypes". The Test Neuron script is described in Appendix of this manual.

MATLAB

The GVEngine.dll can be used from MatLab. A NeuroMem Class and variety of examples are available in the NeuroMem_SDK_4MatLab.

HARDWARE INTERFACE

int Version();

Returns the version of the DLL. The last digit represents the platform 0= Simulation or 1= NeuroStack.

Void SetPlatform(int platform);

Set the hardware platform: 0= Simulation, 1=NeuroStack

int Connect(int DeviceID);

Establishes communication with the NeuroMem device. This function returns 0 if the connection is successful.
In the case of a USB device, DeviceID is the USB device number connected to your host. Default is 0.

int Disconnect();

Closes the communication with the current device.

int CountNeuronsandReset();

Detect the number of neurons available in the connected platform and then clear them all including their registers and memory.

Remark 1: Detecting the neurons requires to commit them all. As a consequence, its execution will overwrite the existing content of the neurons if any if committed.

Remark 2: The execution of this function can take several seconds depending on the size of the chain of CM1K chips plus it erases the contents of the neurons.

Pursuant to the two previous remarks, it is highly recommended to use CountNeuronsandRset only at the launch of an application.

LEARNING AND RECOGNITION FUNCTIONS

void Broadcast(byte vector[], int length)

Broadcast the input vector composed of “length” components to the neurons. The value length is cropped to 256 of necessary. If the length of the vector is L, this function executes L-1 Write COMP followed by a single Write LCOMP. It can be used whether you intend to learn or recognize the vector.

Considerations before calling this function:

- Change the Global Context Register (GCR) if the pattern belongs to a context other than the active context

- Change the Network Status Register (NSR) to turn ON/OFF the KNN classifier
- Change the Min Influence Field and Max Influence Field

Considerations after calling this function:
- Read the Network Status Register to find out if the vector is recognized or not, with uncertainty or not
- Write the Category register to learn the vector
- Read the Distance, followed by Category to classify the vector

void BroadcastSR(byte vector[], int length)

The intend of this function is to broadcast the input vector composed of "length" components only when the network is already set to the Save-and-Restore mode. The value length is cropped to 256 of necessary. If the length of the vector is L, this function executes L consecutive Write COMP. It must be followed to a Write CAT to focus on the next neuron of the chain.

int Learn(unsigned char vector[], int length, int category);

Learn the input vector composed of "length" components as belonging to the "category". The value length is cropped to 256 of necessary. The function returns the number of committed neurons. Category can range between 1 to 32766. A category of 0 can be used to teach a counter example.

Considerations before calling this function:
- Change the Global Context Register to match the context represented by the input vector
- Change the Network Status Register (NSR) to turn ON/OFF the KNN classifier, or to turn ON/OFF the Save-and-Restore mode.
- Change the Min Influence Field and Max Influence Field

int Recognize(unsigned char vector[], int length, int K, int distance[], int category[], int Nid[]);

Recognize the input vector composed of "length" components and read the response of the K first firing neurons reporting their *Distance*, *Category* and *Identifier* register. The value length is cropped to 256 of necessary. The response of the firing neurons are automatically read in order of decreasing confidence. The function returns the classification status (0=unknown, 8=identified, 4=uncertain).

Considerations before calling this function:
- Change the Global Context Register to match the context represented by the input vector
- Change the Network Status Register (NSR) to turn ON/OFF the KNN classifier, or to turn ON/OFF the Save-and-Restore mode.

Remark:
- This function returns exhaustive results. If an application does not require all this information, you may want to use the Broadcast function and only read the registers you need for optimization purposes.

void ReadLastVector(unsigned char vector[]);

Retrieve the last vector which was broadcasted to the neurons and is still held in the Ready-To-Learn neuron.

LOADING AND RETRIEVAL OF NEURONS' CONTENT

int CommittedNeurons();

Report the number of committed neurons.

A value of 0x00FFFFFF (16,777,215) indicates that the network is full and not that over 16 millions of neurons are committed.

Remark: The NCOUNT register of the CM1K chip is a single clock cycle instruction returning the number of committed neurons on a 16-bit value. It is perfectly appropriate to read this register if your hardware platform does not include a chain of more than 0xFFFF (65,535 neurons). On the other end, if your hardware platform is composed of a chain of more than 65,353 neurons or might expand to such capacity in the future, it is recommended to use the CommittedNeurons function from the beginning instead of a Read NCOUNT. The CommittedNeurons takes four clock cycles.

int ClearNeurons();

Clear the entire contents of all the neurons including registers and memory cells. The function returns the number of committed neurons which should be equal to zero.

void ReadNeuron(int neuronID, unsigned char model[], int context, int aif, int minif, int category);

Output the content of the IDth neuron in the chain including:
Model[]= 256 bytes of memory*
Context= Neuron context register
Aif= Active Influence Field register
Minif= Minimum Influence Field register
Cat= Category register

NeuronID can range from 1 to the number of committed neurons. If neuronID greater, the function returns the contents of the RTL (Ready-To-Learn) neuron.

*Note that if an application is dealing with vectors of a length L lesser than 256, only the first L values of the model[] are significant. If the ClearNeurons function was executed prior to learning the first vector, the remaining components from L+1 to 256th will be equal to 0, otherwise they will report values loaded at an earlier stage into the neurons.

int ReadNeurons(unsigned char *neurons, int ncount);

Read the first "ncount" neurons of the chain. Their content is returned in the format of an array of "ncount" times 264 bytes. The function returns the actual number of neurons read by the function, which can be less than the "ncount" input value. In such case, the neurons array of padded with zeros.

The content of a neuron is formatted in a 264-byte array as follow:

- Byte 0= 0 (discarded)

- Byte 1= NCR, Neuron Context Register
- Byte 2-258= 256 components of the pattern vector*
- Byte 259-260: AIF, Active Influence Field
- Byte 261-262: MINIF, Minimum Influence Field
- Byte 263-264: CAT, Category

*Note that if an application is dealing with vectors of a length L lesser than 256, only the first L values of the model[] are significant. If the ClearNeurons function was executed prior to learning the first vector, the remaining components from L+1 to 256th will be equal to 0, otherwise they will report values loaded at an earlier stage into the neurons.

int WriteNeurons(unsigned char *neurons, int ncount);

Load the contents of the neurons from an array of "ncount" times 264 bytes.

The content of a neuron is formatted in a 264-byte array as follow:

- Byte 0= 0 (discarded)
- Byte 1= NCR, Neuron Context Register
- Byte 2-258= 256 components of the pattern vector*
- Byte 259-260: AIF, Active Influence Field
- Byte 261-262: MINIF, Minimum Influence Field
- Byte 263-264: CAT, Category

int SaveNeurons(char *filename);

Save the contents of all the neurons to a file. The function returns the number of committed neurons saved to the file.

int LoadNeurons(char *filename);

Load the contents of all the neurons from a file. The function returns the number of committed neurons loaded from the file.

VECTOR BATCH PROCESSING

int LearnVectors(unsigned char *vectors, int vectNbr, int vectLen, int *categories, bool ResetBeforeLearning, bool iterative);

Learn an array of vectors with their asscoiated categories. The "vectNbr" vectors are all composed of the same number of components "vectLen". It returns the total number of committed neurons.

- vectors= byte Array with a size vectNbr * vectLen
- categories= int Array with a size of vectNbr

The flag **ResetBeforeLearning** allows to reset the network prior to learning. Note that this reset is a simple Write FORGET which takes a simple clock cycle, and not a ClearNeurons which erases the content of the neurons'

memory but takes 258clock cycles. If ResetBeforeLearning is set to 1, the function preserves the value of the minimum and maximum Influence fields.

The flag **Iterative** executes the learning of the vectors repeatedly until the number of committed neurons remains steady between two passes. This option produces a knowledge which is invariant to the order of the vectors, but might take multiple iterations and consequently some additional time.

Considerations before calling this function:
- Clear the neurons to pad their memory with zero in case vectLen is less than 256
- Set the NSR register to 16 or 0
    - If NSR=16, the vectors are loaded in Save and Restore mode
    - If NSR=0, the vectors are learned using the RBF classifier
- Change the Global Context Register to match the context represented by the input vectors
- Change the Maxif : The Maxif is used when learning in normal mode (NSR=0). The higher the Maxif, the faster the teaching, but the higher the probability of false positive
- Change the Minif : The Minif is used when learning in normal mode (NSR=0). The higher the Minif, the more degenerated neurons and zones of uncertainty.

int BuildCodebook(unsigned char *vectors, int vectNbr, int vectLen, int CatAllocMode, int InitialCategory, bool WithoutUncertainty);

Learn an array of vectors with unsupervised allocation of categories according to the selected method. The “vectNbr” vectors are all composed of the same number of components “vectLen”. This function does not clear the existing knowledge. It returns the total number of committed neurons.

- vectors= byte Array with a size vectNbr * vectLen
- CatAllocMode: Defines which category to assign to a vector which is not recognized by the currently committed neurons:
    - 0: constant value
    - 1: auto-increment by 1
    - 2: maximum delta between the vectLen components
    - 3: average value of the vectLen components.
- InitialCategory
    - Initial value for the CatAllocMode 0 and 1
- waiveUncertainty
    - If set to 1, this flag forces the learning of a vector which is recognized by the currently committed neurons but with an uncertainty or disagreement between at least two firing neurons. This will force the reduction of the influence fields of some committed neurons

Considerations before calling this function:
- Clear the neurons to pad their memory with zero in case vectLen is less than 256
- Set the NSR register to 16 or 0
    - If NSR=16, the vectors are loaded in Save and Restore mode
    - If NSR=0, the vectors are learned using the RBF classifier
- Change the Global Context Register to match the context represented by the input vectors
- Change the Maxif : The Maxif is used when learning in normal mode (NSR=0). The higher the Maxif, the faster the teaching, but the higher the probability of false positive
- Change the Minif : The Minif is used when learning in normal mode (NSR=0). The higher the Minif, the more degenerated neurons and zones of uncertainty.

int Clusterize(unsigned char *vectors, int vectNbr, int vectLen, int CatAllocMode);

Learn the input vectors assigning their categories autonomously according to the selected method. This function clears the existing knowledge prior to learning. It returns the total number of committed neurons.

It is slightly different from the BuildCodebook in the sense that an input vector is considered as a novelty if (a) it is unknown, or (b) it is known but at a distance greater than MINIF from an existing model. The value of the MINIF register set before the execution of the command forces a minimum spacing between the models.

- vectors= byte Array with a size vectNbr * vectLen
- CatAllocMode: Defines which category to assign to a vector which is not recognized by the currently committed neurons:
    - 1: auto-increment by 1
    - 2: maximum delta between the vectLen components
    - 3: average value of the vectLen components.

Considerations before calling this function:
- Save the current knowledge which will be erased otherwise.
- Clear the neurons to pad their memory with zero in case vectLen is less than 256
- Set the NSR register to 16 or 0
    - If NSR=16, the vectors are loaded in Save and Restore mode
    - If NSR=0, the vectors are learned using the RBF classifier
- Change the Global Context Register
- Change the Maxif : The Maxif is used when learning in normal mode (NSR=0). The higher the Maxif, the faster the teaching, but the higher the probability of false positive
- Change the Minif : The Minif is used when learning in normal mode (NSR=0). The higher the Minif, the more degenerated neurons and zones of uncertainty.

void RecognizeVectors(unsigned char *vectors, int vectNbr, int vectLen, int K, int *distances, int *categories, int *nids);

Recognize an array of vectors and return the response of the first K firing neurons in the 3 arrays distances, categories and nids. The “vectNbr” vectors are all composed of the same number of components “vectLen”.

- vectors= byte Array with a size vectNbr * vectLen
- distances= int Array of size vectNbr*K reporting for each vector the distances of the K top firing neurons or 0xFFFF if less than K neurons fire.
- categories= int Array of size vectNbr*K reporting for each vector the categories of the K top firing neurons or 0xFFFF if less than K neurons fire.
- nids= int Array of size vectNbr*K reporting for each vector the identifiers of the K top firing neurons or 0xFFFF if less than K neurons fire.

Considerations before calling this function:
- Change the Global Context Register to match the context represented by the input vectors

int MatchVectors(unsigned char *vectorsRef, int vectRefNbr, unsigned char *vectors, int vectNbr, int vectLen, int maxDistance, int *indexPairs, int *matchDistances);

Match two sets of vectors of the same length and produces the pairs of matching indexes and their distances. This function clears the neurons prior to execution. It returns the number of pairs found.

- vectorsRef= byte Array with a size vectRefNbr * vectLen
- vectors= byte Array with a size vectNbr * vectLen
- maxDistance= distance not to exceed to consider a match between a *vector* and its associated *vectorRef*
- indexPairs= Array[matches_found,2]
    - the first value of the pair is the index of *vector*
    - the second value of the pair is the index of the closest *vectorRef*
- matchDistances[matches_found]
    - distance between the two vectors of an indexPairs

Considerations before calling this function:
- Save the current knowledge which will be erased otherwise.

GV COMMUNICATION PROTOCOL

The NeuroMem hardware platforms use a same protocol implemented in the SDK:

1 byte	4 bytes	3 bytes	N words
Device ID if applicable Default =1	bit[31]= RW command 0, Read 1, Write bit[30-24]= Module ID bit[23-0]= Register	Datalength expressed in bytes Default = 1	Array of Datalength bytes Sent if bit[31]=1 Received if bit[31]=0

A single read or write command takes 10 bytes which can be translated with the following sequences:

Write(byte **ModuleID**, byte **Register**, int **Data**)
Read(byte **ModuleID**, byte **Register**, int **Data**)

1 byte	1 byte	2 bytes	1 byte	3 bytes	2 bytes
x01	bit[7]= 1 for Write, 0 for Read bit[6:0]= **Module ID**	0x0000	**Register**	x000002	**Data**

A multiple read or write command takes 8+N*2 bytes.

WriteAddress(long **Address**, long **Length_inBytes**, byte[] **Data**)
ReadAddress(long **Address**, long **Length_inBytes**, byte[] **Data**)

1 byte	4 byte	3 bytes	**Length_inBytes**
x01	bit[31]= 1 for Write, 0 for Read bit[30:0]= **Address**	**Length_inBytes**	**Data**

All platfoms have in common that their chain of CM1Ks is referred to as the *module* number 1. Depending on the platform, additional modules may be accessible such as a bank of memory, a sensor, etc. For more information on these modules refer to the technical manual of the hardware.

int Write(unsigned char module, unsigned char reg, int data);

Write the value *data* to the register *reg* of the module *module*.
Example: Write(1, 6, 3), set the minimum influence field of the network to the value 3.

int Read(unsigned char module, unsigned char reg);

Return the value *of* the register *reg* of the module *module*.
Example: Read(1, 15) returns the number of committed neurons

int Write_Addr(int addr, int length_inByte, unsigned char data[]);

Write the values of the byte array *data* to the address *addr* of the hardware platform.

Remark: The internal communication protocol of the various NeuroMem platforms reads and writes integer values to the registers of the modules. This means that the data array is actually read two bytes at a time and each two bytes are assembled as the upper and lower bytes of a register value.

- data= byte array[length_inByte]
- length_inByte, must be an even number of bytes (see remark above)
- addr= 4 bytes value formatted as follows:*[module*, 0, 0, *reg*]

Warning: In the case of the NeuroMem module 1, the usage of this function only makes sense for the CM_COMP register when a vector is broadcasted to the neurons or the content of the neurons is restored. However, please note the following:
- If the vector to broadcast to the neurons has N components, use Write_Addr to submit only the first N-1 component. Indeed the last component has to be written to the CM_LCOMP register.
- The N-1 components have to be formatted to a 2*(N-1) data array such that every odd byte is equal to 0 and even byte is equal to a vector component.

Example:
Vector=1,2,3,4 ➔ data=0,1,0,2,0,3,0,4
Write(0x01000001, 4, data) writes the components 1,2,3 and 4 to the CM_COMP register.

int Read_Addr(int addr, int length_inByte, unsigned char data[]);

Read the values at the address *addr* of the hardware platform and store them to the byte array *data*.

Remark: The internal communication protocol of the various NeuroMem platforms reads and writes integer values to the registers of the modules. This means that the data array is actually filled two bytes at a time and each two bytes are assembled as the upper and lower bytes of a register value.

- data= byte array[length_inByte]
- length_inByte, must be an even number of bytes (see remark above)
- addr= 4 bytes value formatted as follows:*[module*, 0, 0, *reg*]

Warning: In the case of the NeuroMem module 1, the usage of this function only makes sense for the CM_COMP register in order to read the memory of a neuron. However, please note the following:

- Reading the N components of a neuron requires that you size the data array to 2*(N-1) bytes. Every odd byte is equal to 0 and even byte is equal to a neuron component.

Example:
Vector=1,2,3,4 ➔ data=0,1,0,2,0,3,0,4
Write(0x01000001, 4, data) writes the components 1,2,3 and 4 to the CM_COMP register.

From Address Map to (Module,Register)

Address mapping may defer between hardware platforms and below is the example of the NeuroStack board.

Address Range	**Module=** Address[30-24]	**Functionality defined by registers =** Address[23:8]
0x01000000 0x0100001F	NeuroMem 0x01 (d01)	Access to the registers of the CM1K chips whether limited to the chain of 4 chips on a single board or a chain of N*4 chips on a stack of N boards.
0x02000000 0x0200001F	Board settings 0x02 (d02)	Access to the settings of the board connected to the host including but not limited to a version number, a clock counter, etc.
0x03000000 0x03FFFFFF	MRAM 0x03 (d03)	Access to the bank of MRAM of the board connected to the host.
0x04000000 0x0400001F	Recognition Engine 0x04 (d04)	Access to the recognition engine programmed in the FPGA.

For information about the NeuroMem module and its registers, refer to the CM1K API manual and CM1K Hardware Manual. For information about the modules and registers specific to your hardware platform, refer to its hardware manual.

CYCLE ACCURATE TIMINGS

The API features a cycle accurate counter to evaluate the latency of the operations executed by the NeuroMem network. Multiplying this counter by the duration of a clock cycle gives time spent by the entire network to execute a given function. Note that this does not include the data transfer time between the network and the host.

Please note that enabling the cycle accurate counter imposes some overhead operations on the basic RTL functions and will consequently slow down the execution of the functions if timed by the host processor.

void EnableClockCounter()

Turn ON the usage of the cycle accurate counter and reset the counter value.

void DisableClockCounter()

Turn OFF the usage of the cycle accurate counter.

int ReadClockCounter()

Read the current value of the counter.

void ClearClockCounter()

Clear the counter.

Example using clock counter

EnableClockCounter	
Read CM_DIST	18 cycles
Read CM_CAT	19 cycles if ID_ is high, 3 cycles otherwise
ReadClockCounter	returns (37)

EXAMPLE SCRIPT

The following script is intended to illustrate the behavior of the neurons during learning and recognition operations. It is supplied in a variety of programming language including C/C++. C#, MatLab, Python, Arduino and more.

Step 1: Learn

Learn vector1 [11,11,11,11] $_{(1)}$ with category 55
Learn vector2 [15,15,15,15] with category 33$_{(2)}$
Leanr vector3 [20,20,20,20] with category 100

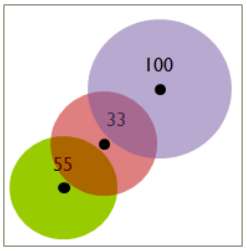
(1) The learned vectors are purposely set to arrays of constant values so their representation and relationship are easy to understand. The distance between two "flat" vectors is indeed the difference between their constant value times their number of components. For example the distance between [11,11,11,11] and [15,15,15,15] is equal to 4 * 4. This simple arithmetic makes it easy to understand the different cases of recognition illustrated in this test script.

(2) The category of the second neuron is purposely set to a lesser value than the category of the first neuron to verify that if both neurons fire with a same distance, the category of the neuron on the 2nd chip is still the first the be read out

Fig1 is a representation of the decision space modeled by the 3 neurons where Neuron1 is shown in red, Neuron2 in green and Neuron3 in blue. In the follwoing 2D graph, we limit the length of the models to 2 components instead of 4, so they can be positioned in an (X,Y) plot. X=1st component and Y=Last component, and a surrounding diamond shape with the side equal to their influence field.

Committed neurons= 3

NeuronID=1	Components=11, 11, 11, 11,	AIF=16,	CAT=55
NeuronID=2	Components=15, 15, 15, 15,	AIF=16,	CAT=33
NeuronID=3	Components=20, 20, 20, 20,	AIF=20,	CAT=100

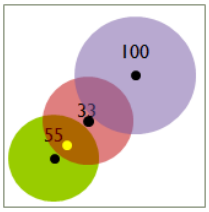


The influence fields of Neuron#0 and Neuron#1 overlap, as well as Neuron#1 and Neuron#2 overlap but differently since their distances from one another are different.

Step2: Recognition

The vectors submitted for recognition are selected purposely to illustrate cases of positive recognition with or without uncertainty, as well as cases of non recognition. The program reads the responses of all the firing neurons, which is until the distance register returns a value 0xFFFF or 65553.

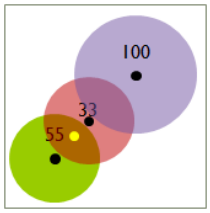
CASE OF UNCERTAINTY, CLOSER TO NEURON#1



Vector=12, 12, 12, 12
Neuron 1 and 2 fire. Vector is closer to Neuron1

Response#1	Distance= 4	Category= 55	NeuronID= 1
Response#2	Distance= 12	Category= 33	NeuronID= 2
Response#3	Distance= 65535	Category= 65535	NeuronID= 65535

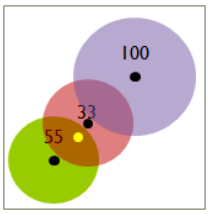
CASE OF UNCERTAINTY, EQUI-DISTANT TO NEURON#1 AND NEURON#2



Vector=13, 13, 13, 13,
Neuron 1 and 2 fire. Vector is equi-distant to both neurons

Response#1	Distance= 8	Category= 33	NeuronID= 2
Response#2	Distance= 8	Category= 55	NeuronID= 1
Response#3	Distance= 65535	Category= 65535	NeuronID= 65535

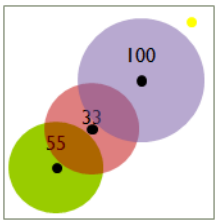
CASE OF UNCERTAINTY, CLOSER TO NEURON#2



Vector=14, 14, 14, 14,
Neuron 1 and 2 fire. Vector is closer to Neuron2

Response#1	Distance= 4	Category= 33	NeuronID= 2
Response#2	Distance= 12	Category= 55	NeuronID= 1
Response#3	Distance= 65535	Category= 65535	NeuronID= 65535

CASE OF UNKNOWN

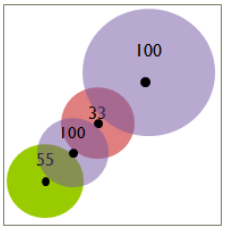


Vector=30, 30, 30, 30,
No neuron fire
Response#1 Distance= 65535 Category= 65535 NeuronID= 65535

Step3: Adaptive learning

Learning a new vector to illustrate the reduction of neurons'AIFs.

Learn vector[13,13,13,13] with category 100. This vector is equidistant to both Neuron1 and Neuron2. Learning it as a new category, will force Neuron1 and 2 to shrink from thei AIF=16 to an AIF=8 in order to make room for a new neuron which will hold the new model and its category.



Committed neurons= 4

NeuronID=1	Components=11, 11, 11, 11,	AIF=8,	CAT=55
NeuronID=2	Components=15, 15, 15, 15,	AIF=8	CAT=33
NeuronID=3	Components=20, 20, 20, 20,	AIF=20	CAT=100
NeuronID=4	Components=13, 13, 13, 13,	AIF=8,	CAT=100

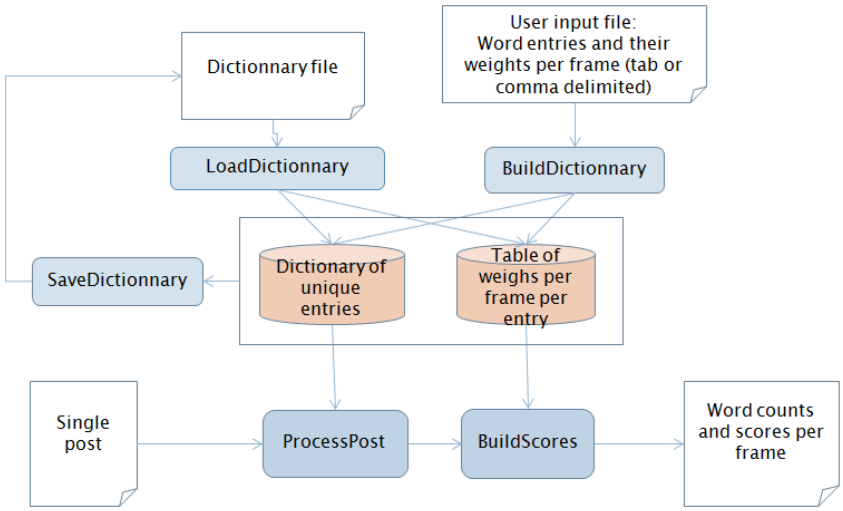
Note that if the vector to learn was [12,12,12,12], Neuron1 would shrink to 4 and Neuron2 to 12.

SUPPLEMENT: TEXT ANALYTICS FUNCTIONS

The text analytics functions are ready to use for keyword spotting and latent sentiment analytics provided that you have a user input file with a compatible format.

Examples and ready-to-use datasets are included in the SDK. The dictionary of words with their associated weights per sentiment come from DepecheMood “a Lexicon for Emotion Analysis from Crowd-Annotated News”.

The Text Analytics examples in C++ and Python analyze posts saved to file. The Tweet Analytics example in Python retrieves live tweets.



Remarks about patterns composed of ASCII characters

Remark#0:
These patterns are learned to detect exact matches. Because the length of the entries varies, the learned vectors must be padded with “space” characters to reach at least a length L which is the length of the longest entry if known in advance, or 256 otherwise.

Remark#1:
It is possible that the L1 distance between two strings of ASCII characters turns equal to zero through unlucky substitutions, even though the strings are not identical. The use the Norm LSUP is necessary to avoid this confusion.

Remark#2:
If an entry representing a plural such as “animals” has been learned, the neuron holding this pattern will fire if the entry representing the singular “animal” is submitted and consequently “animal” will not be learned.

- ⇨ The addition of a Space char at the end of each entry is an easy way to remedy this unwanted behavior of the neurons

int BuildDictionnary(char *Filename, int NbrOfFrames, char *delimiter)

This function requires an input file with the following format (delimited with comma, tab or other separator)

- field 0 = Word (up to 256 char)

- field 1 to NbOfFrames = weight of Word per frame. A frame can represent a topic, a sentiment or else. A same word can have different weighs for different frame.

The function erases any previous knowledge and builds two data structures:
- a dictionnary of non-duplicated reference words
- a table of the weighs associated to these words per frame

```
Example of input file with 8 frames
```

Lemma#PoS	AFRAID	AMUSED	ANGRY	ANNOYED	DONT_CARE	HAPPY	INSPIRED	SAD
Abandon	0.1874	0.15971	0.11084	0.07673	0.12806	0.12490	0.1073	0.10491
Abandoned	0.10648	0.14124	0.11563	0.1323	0.13463	0.09281	0.11450	0.16236

```
Remark: This function builds a knowledge of unigrams, which are defined as words
separated by the following characters: “ !?()[]{}/.:;,”
```

void SaveDictionnary(char *Filename)

Save the dictionary to file including a header composed of the variable frame number, word number and max word length, followed by the weightTable of size (woNbr*frNbr), followed by the neurons' content.

int LoadDictionnary(char *Filename)

Load the dictionnary from file including a header composed of the variable frame number, word number, and max word length, followed by the weightTable of size (woNbr*frNbr), followed by the neurons' content.

Remark: this format is generated by the SaveDictionnary command.

void ProcessPost(char* post, int* woCount)

Process a single post and calculate its histogram of recognized words. Scores can be retrieved with the BuildScores function.
- woCount= the hitcount per words of the the dictionnary (woNbr*1)

void BuildScores(float* scoreTable)

Retursn the scores of the last processed post
- Scores= score per sentiment (frNbr*1) (i.e. hitcount of the words in dictionnary multiplied by their weights for each sentiment)

void DisplayDictionnary()

Display the dictionary of reference words and their weights in the various frames (sentiments) on the console.